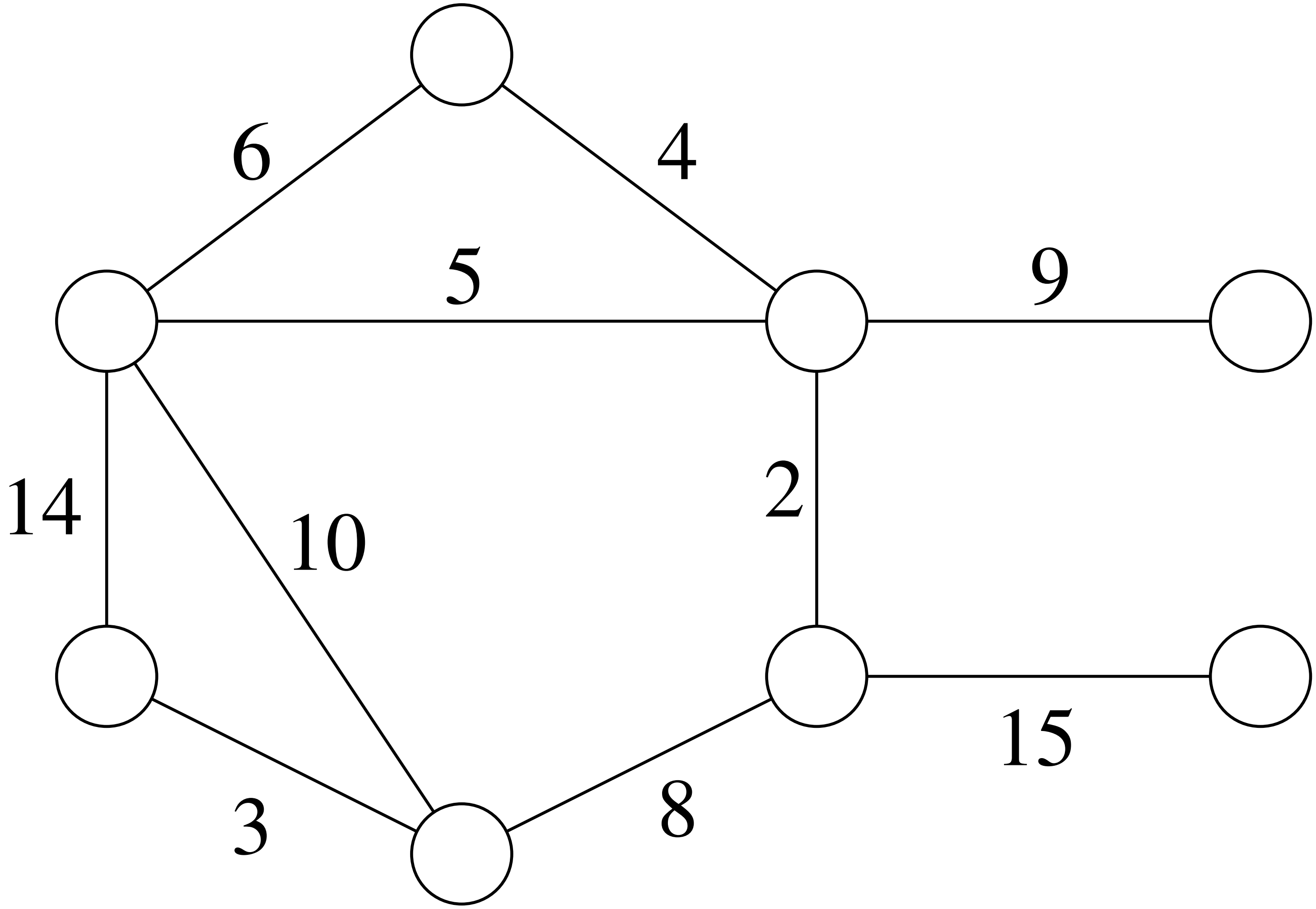
Minimum spanning trees

One of the most famous greedy algorithms
(actually rather *family* of greedy algorithms).

- Given undirected graph $G = (V, E)$, connected
- Weight function $w : E \to \mathbb{R}$
- For simplicity, all edge weights distinct
- Spanning tree: tree that connects all vertices, hence $n = |V|$ vertices and $n - 1$ edges
- MST $T : w(T) = \sum_{(u,v) \in T} w(u, v)$ minimized

What for?

- Chip design
- Communication infrastructure in networks


6
4
5
9
14
10
2
15
3
8

Growing a minimum spanning tree

First, "generic" algorithm. It manages set of edges A, maintains invariant:

Prior to each iteration, A is subset of some MST.

At each step, determine edge (u, v) that can be added to A, i.e. **without violating invariant**, i.e., $A \cup \{(u, v)\}$ is also subset of some MST. We then call (u, v) a **safe edge**.

$A \leftarrow \emptyset$
while A does not form a spanning tree **do**
    find an edge (u, v) that is safe for A
    $A \leftarrow A \cup \{(u, v)\}$
end while

We use invariant as follows:

Initialization. After line 1, A triv. satisfies invariant.

Maintenance. Loop in lines 2–5 maintains invariant by adding only safe edges.

Termination. All edges added to A are in a MST, so A must be MST.

How to recognize safe edges?

Definitions

1. A *cut* $(S, V - S)$ of an undir. graph $G = (V, E)$ is a partition of V.
2. An edge (u, v) *crosses* cut $(S, V - S)$ if one endpoint is in S, the other one in $V - S$.
3. A cut *respects* a set $A \subseteq E$ if no edge in A crosses the cut.
4. An edge is a *light edge* crossing a cut if its weight is the minimum of any edge crossing the cut.

Theorem 1. Let A be a subset of E that is included in some MST for G, let $(S, V - S)$ be any cut of G that respects A, let (u, v) be a light edge crossing $(S, V - S)$.

Then, (u, v) is safe for A.

Proof.

- let T be a MST that includes A and assume T does not include (u, v)
- **Goal:** construct MST T' that includes $A \cup \{(u, v)\}$. This shows that (u, v) is safe (by def.)
- $(u, v) \notin T$, so there must be path

$$p = (u = w_1 \rightarrow w_2 \rightarrow \cdots \rightarrow w_k = v)$$

  with $(w_i, w_{i+1}) \in T$ for $1 \leq i < k$

- u and v are on opposite sides of cut $(S, V - S)$, so there must be at least one edge (x, y) of T crossing cut
- (x, y) is not in A because A respects cut
- (x, y) is on unique path from u to v, so removing (x, y) breaks T into two components

- adding (u, v) reconnects them to form new spanning tree $T' = T - \{(x, y)\} \cup \{(u, v)\}$

- (u, v) is light edge crossing $(S, V - S)$, and (x, y) also crosses this cut, therefore $w(u, v) \leq w(x, y)$ and

$$W(T') = w(T) - w(x, y) + w(u, v) \leq W(T)$$

  Hence T' is MST.

- $A \subseteq T$ and $(x, y) \notin A$ (this was because (x, y) *crosses* cut but A *respects* cut), so $A \subseteq T'$

- Since $(u, v) \in T'$, we have $A \cup \{(u, v)\} \subseteq T'$ and (u, v) is safe for A

q.e.d.

We see:

- at any point, graph $G_A = (V, A)$ is a **forest** with components being **trees**
- Any safe edge (u, v) for A connects distinct components of G_A, since $A \cup \{(u, v)\}$ must be acyclic
- main loop is executed $|V| - 1$ times (one iteration for every edge of the resulting MST)

We'll see Kruskal's and Prim's algorithms, they differ in how they specify rules to determine safe edges.

In Kruskal's, A is a **forest**; in Prim's, A is a **single tree** .

The following is going to be used later on.

Corollary. Let A be subset of E that is included in some MST for G, let $C = (V_C, E_C)$ be a connected component (tree) in forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is safe for A.

Proof. The cut $(V_C, V - V_C)$ respects A (A *defines* the components of G_A), and (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A.

Kruskal's algorithm

Kruskal's adds in each step an edge of least possible weight that connects two different trees.

If C_1, C_2 denote the two trees that are connected by (u, v), then since (u, v) must be light edge connecting C_1 to some other tree, the corollary implies that (u, v) is safe for C_1.

Implementation

This particular implementation uses `Disjoint-Set` data structure.

Each set contains vertices in a tree of the current forest.

- `Make-Set`(u) initializes a new set containing just vertex u.

- `Find-Set`(u) returns representative element from set that contains u (so we can check whether two vertices u, v belong to same tree).

- `Union`(u, v) combines two trees (the one containing U with the one containing v).

The Algorithm

Given: graph $G = (V, E)$, weight function w on E

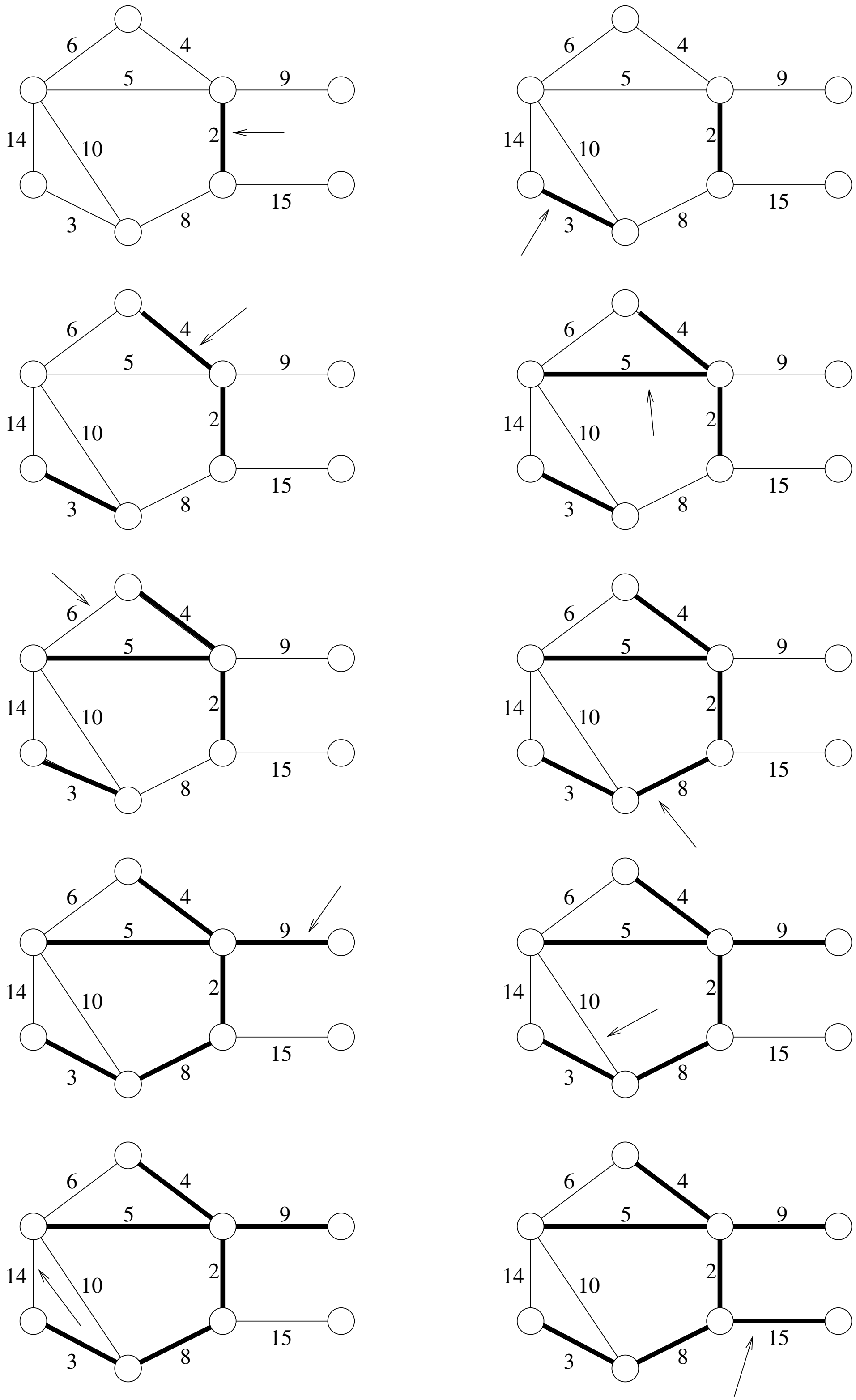
$A \leftarrow \emptyset$
for each vertex $v \in V[G]$ **do**
    `Make-Set`(u)
end for
sort edges of E into nondecr. order by weight w
for each edge $(u, v) \in E$, taken in nondecreasing order by weight w **do**
    if `Find-Set`$(u) \neq$ `Find-Set`(v) **then**
        $A \leftarrow A \cup \{(u, v)\}$
        `Union`(u, v)
    end if
end for
return A

- initializing A takes $O(1)$
- sorting edges takes $O(E \log E)$
- main **for** loop performs $O(E)$ `Find-Set` and `Union` operations; along with $|V|$ `Make-Set` operation, this takes

$$O((V+E)) \cdot O(\log E) = O(E \log V)$$

  (see Section 21.4 in book)

- `Disjoint-Set` operations take $O(E \log V)$
- Total running time of Kruskal's is $O(E \log V)$


6
4
5
9
14
10
2
15
3
8

Prim's algorithm

- A always forms a **single tree** (as opposed to a forest like in Kruskal's).
- Tree start from single (arbitrary) vertex r (root) and grows until it spans all of V.
- At each step, a light edge is added to tree A that connects A to isolated vertex of $G_A = (V, A)$.
- By corollary, this adds only edges safe for A, hence on termination, A is MST.

Implementation

Key is **efficiently selecting new edges**. In this implementation, vertices **not** in the tree reside in min-priority queue Q based on a *key* field:

For $v \in V$, key[v] is minimum weight of any edge connecting v to a vertex in tree A; key[v] $= \infty$ if there is no such edge.

Field $\pi[v]$ names parent of v in tree. During algorithm, A is kept implicitly as

$$A = \{(v, \pi[v]) : \; v \in V - \{r\} - Q\}$$

When algorithm terminates, min-priority queue Q is empty, MST A for G is thus

$$A = \{(v, \pi[v]) : \; v \in V - \{r\}\}$$

Given: graph $G = (V, E)$, weight function w, root vertex $r \in V$

for each $u \in V$ **do**
  key$[u] \leftarrow \infty$
  $\pi[u] \leftarrow$ NIL
end for
key$[r] \leftarrow 0$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
  $u \leftarrow$ `Extract-Min`(Q) {w.r.t. key}
  for each $v \in$ adj$[u]$ **do**
    if $v \in Q$ and $w(u, v) <$ key$[v]$ **then**
      $\pi[v] \leftarrow u$
      key$[v] \leftarrow w(u, v)$
    end if
  end for
end while

Lines 1–6

- set key of each vertex to ∞ (except root r whose key is set to 0 so that it will be processed first)
- set parent of each vertex to NIL
- initialize min-priority queue Q

Algorithm maintains **three-part loop invariant**:

1. $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$
2. Vertices already placed into MST are those in $V - Q$
3. For all $v \in Q$, if $\pi[v] \neq$ NIL, then key$[v] < \infty$ and key$[v]$ is weight of a light edge $(v, \pi[v])$ connecting v to some vertex already placed into MST

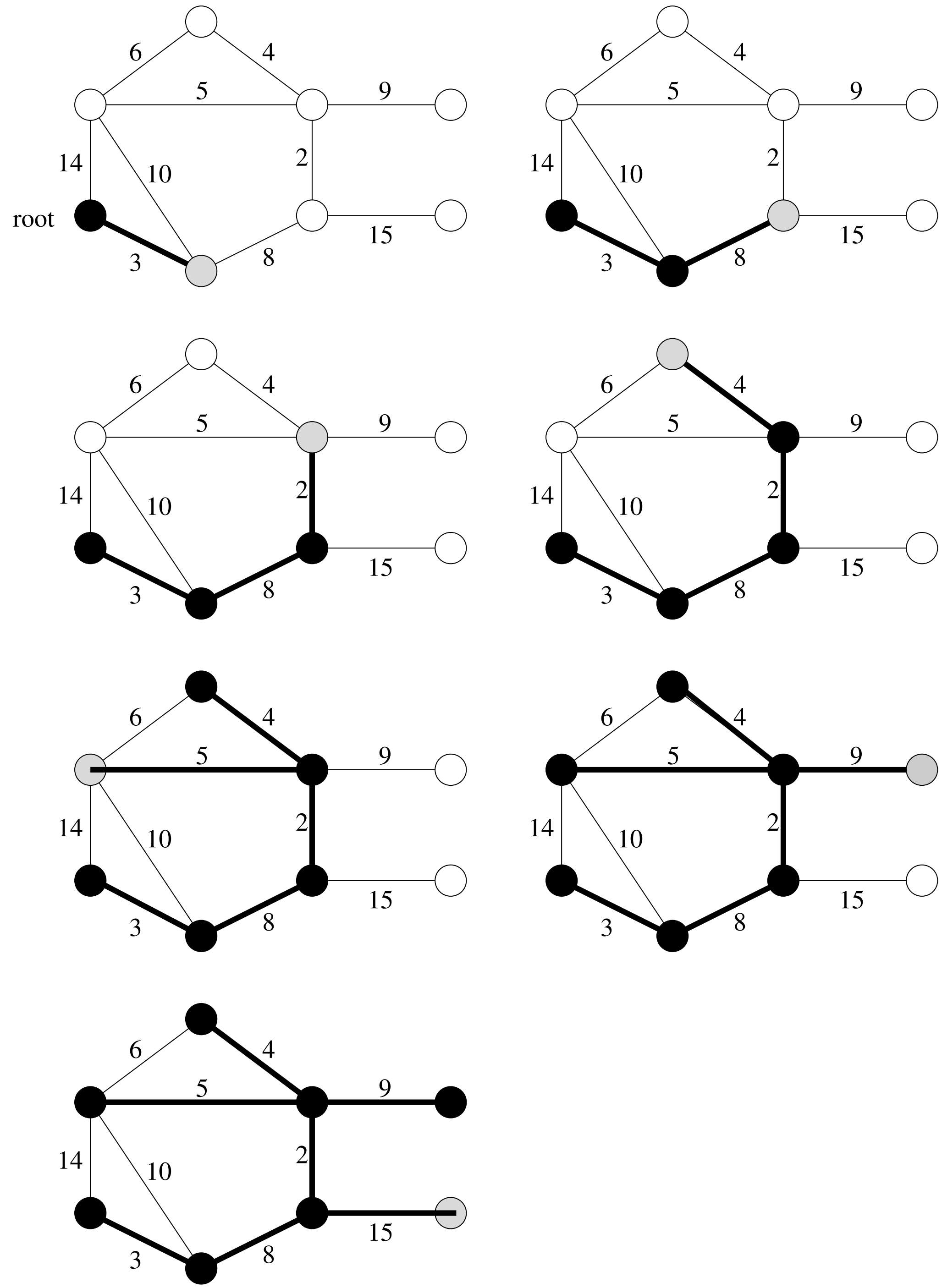
Line 8 identifies $u \in Q$ incident on a light edge crossing cut $(V-Q, Q)$, expect in first iteration, in which $u = r$ due to line 5.
Removing u from Q adds it to set $V-Q$ of vertices in the tree, adding $(u, \pi[u])$ to A.

The **for** loop of lines 9–14 updates the *key* and π fields of every vertex v adjacent to u but **not** in the tree. This maintains third part of loop invariant.

Running time

Depends on how min-priority queue Q is implemented. If as *binary min-heap* (Chapter 6 in book), then

- can use `Build-Min-Heap` for initialization in lines 1–6, time $O(V)$
- body of **while** loop is executed $O(V)$ times, each `Extract-Min` takes $O(\log V)$, hence total time for all calls to `Extract-Min` is $O(V \log V)$
- **for** loop in lines 9–14 is executed $O(E)$ times altogether, since sum of lengths of all adjacency lists is $2|E|$.


root
6
4
5
9
14
10
2
15
3
8